


Improved Parallel Derandomization via Finite Automata with Applications

Jeff Giliberti 

University of Maryland, College Park, MD, USA

David G. Harris 

University of Maryland, College Park, MD, USA

Abstract

A central approach to algorithmic derandomization is the construction of small-support probability distributions that “fool” randomized algorithms, often enabling efficient parallel (NC) implementations. An abstraction of this idea is fooling polynomial-space statistical tests computed via finite automata [Sivakumar STOC’02]; this encompasses a wide range of properties including k -wise independence and sums of random variables.

We present new parallel algorithms to fool finite-state automata, with significantly reduced processor complexity. Briefly, our approach is to iteratively sparsify distributions using a work-efficient lattice rounding routine and maintain accuracy by tracking an aggregate weighted error that is determined by the Lipschitz value of the statistical tests being fooled.

We illustrate with improved applications to the Gale-Berlekamp Switching Game and to approximate MAX-CUT via SDP rounding. These involve further several optimizations, such as the truncation of the state space of the automata and FFT-based convolutions to compute transition probabilities efficiently.

2012 ACM Subject Classification Theory of computation → Parallel algorithms; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Parallel Algorithms, Derandomization, MAX-CUT, Gale-Berlekamp Switching Game

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.70

Related Version *Full Version:* <https://arxiv.org/abs/2411.18028>

Funding *Jeff Giliberti:* JG gratefully acknowledges financial support by the Fulbright U.S. Graduate Student Program, sponsored by the U.S. Department of State and the Italian-American Fulbright Commission. This content does not necessarily represent the views of the Program.

Acknowledgements Thanks to Mohsen Ghaffari and Christoph Grunau for explaining the works [19] and [20].

1 Introduction

A fundamental problem in the theory of computation is to *derandomize* existing randomized algorithms. Such randomized algorithms typically use a large number of independent random bits. One main genre of derandomization is the construction of a probability distribution which is much smaller (of polynomial instead of exponential size), to “fool” the randomized algorithm. That is, the behavior of relevant statistics should be similar when presented with fully-independent random bits vs. bits drawn from a small, carefully-constructed correlated distribution. This probability space can be searched exhaustively, and in parallel, to find a specific input with desired properties.



© Jeff Giliberti and David G. Harris;
licensed under Creative Commons License CC-BY 4.0

33rd Annual European Symposium on Algorithms (ESA 2025).

Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman; Article No. 70; pp. 70:1–70:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A simple and popular example of this technique comes from a probability space with k -wise independence, for constant k [2, 30, 32, 35]. This preserves basic statistical properties such as means and variances [7, 42]. There are many other constructions for more-advanced properties, e.g., near- k -wise independence [3, 5, 14, 17, 38], probability spaces fooling halfspaces or polytopes [33, 41, 23, 22], ordered branching programs [12, 15, 37, 27, 40] etc.

A significant abstraction of these methods is the construction of probability spaces that fool statistical properties (also called “tests”) computed by finite automata [10, 24, 31, 36, 39, 40, 43]. Such tests are ubiquitous in randomized algorithm analysis, encompassing k -wise independence, sums of random variables, and many other properties. For example, they are used in deterministic parallel algorithms for finding balanced independent sets in graphs [26], for applications of the Lovász Local Lemma in combinatorics [25], for covering and packing integer programs [6, 44], for undirected connectivity problems [40], and more.

1.1 Our contribution

We describe new parallel algorithms to fool automata, with significantly reduced processor complexity. The analysis is also simplified because we can cleanly separate out problem-specific optimizations from the general lattice discrepancy problems that are at the core of the algorithm. A summary of our algorithm performance is as follows:

► **Theorem 1 (Simplified).** *Consider a probability space Ω on n binary random variables and a collection of ℓ statistical tests over Ω , each with η possible states and Lipschitz value λ .*

There is a deterministic algorithm to generate a distribution D of size $\varepsilon^{-2} \text{polylog}(n, \eta, \ell, 1/\varepsilon)$ to simultaneously “fool” all statistical tests to absolute weighted error $\lambda\varepsilon$. It uses $\tilde{O}(n\ell\eta\varepsilon^{-2} + n\ell\eta^\omega)$ processors and polylogarithmic time, where ω is the exponent of matrix multiplication.

By way of comparison, the algorithm of [24] would require roughly $O(n^3\ell^2\eta^4\varepsilon^{-2})$ processors to construct a distribution of size $O(\ell\eta^2\varepsilon^{-2})$. Our full results are more general, easily allowing for non-binary alphabets, or more complex automata types; see Theorem 5 for details. We emphasize that the algorithm is still typically more efficient even for applications with no particular Lipschitz bounds.

We illustrate this framework through two prototypical applications: the Gale-Berlekamp Switching Game and SDP rounding for approximate MAX-CUT.

► **Theorem 2 (Gale-Berlekamp Switching Game).** *Given an $n \times n$ matrix A , there is a deterministic parallel algorithm using $\tilde{O}(n^{3.5})$ processors and polylogarithmic time to find $x, y \in \{-1, +1\}^n$ satisfying $\sum_{i,j} A_{i,j}x_iy_j \geq (\sqrt{2/\pi} - o(1))n^{3/2}$.*

It is interesting that the Gale-Berlekamp analysis revolves around *anti-concentration* bounds, which are precisely the opposite of discrepancy-minimization. Our algorithm for this problem beats the $n^{5+o(1)}$ complexity of the optimized algorithm of [24].

► **Theorem 3 (MAX-CUT Approximation).** *Let $\varepsilon > 0$ be an arbitrary constant. Given an n -vertex m -edge graph $G(V, E)$, there is a deterministic parallel algorithm using $\tilde{O}(mn^3)$ processors and polylogarithmic time to find an $\alpha(1 - \varepsilon)$ approximate MAX-CUT of G , where $\alpha \approx 0.878$ is the Goemans-Williamson approximation constant [21].*

The SDP relaxation of MAX-CUT can be approximated in polylogarithmic time and near-linear work by [1], so we will be concerned with rounding the SDP solution. A rounding procedure was presented in [43], which however required a very large number of processors. We drastically improve it, getting closer to the *sequential* $\tilde{O}(n^3)$ runtime of [9].

The complexity bounds in Theorems 2 and 3 do not depend on fast matrix multiplication; the runtimes are valid even using naive multiplication ($\omega = 3$). Although they are still not fully work-efficient, this makes progress toward practical deterministic parallel algorithms.

In a broader perspective, our construction differs in various aspects from classical derandomization via (sequential) PRGs. While our automata framework does not provide a PRG, it seems better suited to efficient parallel derandomization. It is known that, by connections to Boolean circuits [11], a PRG for log space-bounded computation can be simulated in PRAM in polylog time with a polynomial number of processors. However, relying on the equivalence between PRAM and Boolean circuits leads to very high processor complexities (e.g., Nisan's log space-bounded PRG uses at least $\Omega(n^{45})$ processors [39]). Therefore, using known PRG constructions would either require high polynomials (if logspace bounded) or not parallelize well. In addition to providing fast parallelization, our construction returns a distribution with seed length $O(\log 1/\varepsilon + \log \log(n, \eta, \ell))$, which is line with known PRGs.

1.2 Technical approach

Our work follows the same general outline as previous algorithms for fooling automata, built from two main subroutines REDUCE and FOOL. The FOOL algorithm (Section 5) recursively invokes REDUCE (Section 4) to construct a fooling distribution, beginning with single-step automaton transitions and progressively scaling up to multi-step distributions. After $\log_2 n$ levels, the final distribution effectively fools n -step walks in the automata.

We introduce several novelties such as a work efficient REDUCE subroutine and an analysis of FOOL based on a different measure of error. Here, we attempt here to provide some high-level intuition on the technical content of this paper.

Distribution error tracking. Prior work on automata fooling [24, 36] considered a notion of unweighted absolute error, where the goal was to fool all pairs of (start, end) states. We replace this with an *aggregated* notion: the weight of a final state corresponds to the value of the associated final outcome (e.g., the final count for a counter automaton). The value of an intermediate state is then the expected weight of the final state under a random suffix. This is similar to a scheme used in the context of pseudorandom generators and branching programs [12, 15], where it is taken advantage of sub-maximal influence on the final expected value. The processor complexity is determined by the *Lipschitz value* of each state: how much the expected final weight changes with a single step of the automaton.

REDUCE subroutine. This subroutine takes as input a distribution $E = D^1 \times D^2$ over automata walks (drivestreams) with a weight function w ; it outputs a refined distribution D with significantly smaller support while maintaining a weighted measure of closeness to E . This leverages the connection between automata fooling and lattice approximation noted by [36] and applies the work-efficient lattice approximation of [19]. Our key approach to reduce the processor complexity is to sparsify E – without materializing it – in $\log_2 |E|$ steps.

In each step, we model the problem of rounding the i^{th} bit of the elements in D as a lattice-discrepancy problem, which is then solved by invoking [19]. This step resembles the PRG of INW [28]; there, one replaces the concatenation of two independent random walks with correlated random walks, using the derandomized square instead of the algorithm of [19].

Further optimizations. In both of these applications, and many others, the relevant automata are counters which track the running sum of certain statistics. Instead of keeping track of these exactly, we truncate the counters to within a few standard deviations of their means.

This reduces the state space of the automata by roughly a square-root factor. Doing this while also preserving relevant Lipschitz properties of the automata is technically challenging and requires significant analysis. This optimization will be discussed in Section 6.

The approximate MAX-CUT application (Section 7.2) requires a few additional problem-specific optimizations and arguments. In particular, to simulate the SDP rounding procedure efficiently, we (i) work with discretized truncated Gaussians and quantized counters, (ii) use a pessimistic estimator with a better Lipschitz constant, and (iii) we compute the transition matrices via FFT's to exploit their convolution structure. We believe that these optimizations may generalize to other (SDP) rounding procedures and other settings as well.

2 Preliminaries

For an input of size N , our goal is to develop deterministic PRAM algorithms with $\text{poly}(N)$ processor complexity and $\text{polylog}(N)$ runtime. Unless stated otherwise, we assume that all relevant algorithms run in *deterministic polylogarithmic time* as a function of their processor and input complexities. For processor complexity and other parameters, we use \tilde{O} notation: we say that $f(x) \leq \tilde{O}(g(x))$ if $f(x) \leq g(x) \cdot \text{polylog}(N, g(x))$ processors, where N is the size of all algorithm inputs. Throughout, we use \log for logarithm in base e and \lg for base 2. We write $\lceil x \rceil$ for rounding to the nearest integer.

Throughout, proofs that are omitted (or sketched) appear in the full version.

2.1 Basic definitions for automata

The underlying probability space Ω is defined by drawing a sequence $\vec{r} = (r_0, \dots, r_{n-1})$, where each r_t is independently drawn from a distribution Ω_t over an arbitrary alphabet. We consider an automaton F with a state space S . At each timestep $t = 0, 1, \dots, n-1$, the automaton in state $s \in S$ receives an input r_t and transitions to state $F(r_t, s)$.

For times t, t' , we define $\Omega_{t,t'} = \Omega_t \times \Omega_{t+1} \times \dots \times \Omega_{t'-1}$. In this context, we call $h = t' - t$ the *horizon* and the pair (t, h) as the *window*. We refer to a vector $\vec{r} = (r_t, r_{t+1}, \dots, r_{t'-1}) \in \Omega_{t,t'}$ as a *drivestream*. We define $F(\vec{r}, s)$ to be the result of transiting from time t to t' under \vec{r} . We write $\vec{r} \in \Omega_{t,t'}$ to indicate that each $r_j : j = t, \dots, t' - 1$ has non-zero probability in Ω_j .

For a distribution D on drivestreams, we denote the *transition matrix* as T_D , i.e. $T(s, s')$ is the probability of transiting from state s to s' for a drivestream $\vec{r} \sim D$. For brevity, we also write $T_{t,t'} := T_{\Omega_{t,t'}}$ and, for any weight function $w : S \rightarrow \mathbb{R}$, we also define

$$T_D(s, w) = \sum_{s' \in S} T_D(s, s') w(s').$$

Throughout, we define $\eta = |S|$ and $\sigma = \sum_{t=0}^{n-1} |\Omega_t|$ where $|\Omega_t|$ is the size of Ω_t .

Our goal is to find a polynomial-size distribution D on drivestreams that “fools” the automaton. That is, the behavior of the automaton given a drivestream $r \sim D$ should be similar to its behavior when given a drivestream $r \sim \Omega$. We will follow a strategy of [12] and measure error via a weight function $W : S \rightarrow \mathbb{R}$ over final states. It will be necessary to measure the “smoothness” of W as a function of the drivestream values. Formally, we consider two, closely related, notions:

► **Definition 4** (Lipschitz value/confusion of a weight functions). *Let $W : S \rightarrow \mathbb{R}$ be a weight function for automaton F .*

■ *The Lipschitz value at state s and time t is defined by*

$$\mathfrak{L}(s, t) = \max_{\vec{r}_1, \vec{r}_2} |W(F(\vec{r}_1, s)) - W(F(\vec{r}_2, s))|$$

where the maximum is over $\vec{r}^1, \vec{r}^2 \in \Omega_{t,n}$ which differ in only the t^{th} coordinate.

■ The confusion at state s and time t is defined by

$$\mathfrak{C}(s, t) = \max_{r_1, r_2 \in \Omega_t} |T_{t,n}(F(r_1, s), W) - T_{t,n}(F(r_2, s), W)|.$$

Colloquially, these are the maximum change to *actual weight* or the *expected weight* of the final state due to changing drivestream entry t . Note that $\mathfrak{C}(s, t) \leq \mathfrak{L}(s, t)$.

We define the *total variability* $\mathfrak{V}(s)$ of a starting state $s \in S$ as:

$$\mathfrak{V}(s) := \sum_{t=0}^{n-1} \max_{\vec{r} \in \Omega_{0,t}} \mathfrak{C}(F(\vec{r}, s), t)$$

With this rather technical definition, we can state our algorithm concretely:

► **Theorem 5.** *The algorithm FOOL takes as input a parameter $\varepsilon > 0$, and produces a distribution D with*

$$|T_D(s, W) - T_\Omega(s, W)| \leq \varepsilon \mathfrak{V}(s) \quad \text{for all states } s.$$

The distribution D has size $\varepsilon^{-2} \text{polylog}(n, \eta, \sigma, 1/\varepsilon)$. The cost of FOOL is $\tilde{O}(n\eta/\varepsilon^2 + \eta\sigma)$ processors, plus the cost of computing the expectations $T_{t,n}(s, W)$ for states $s \in S$ and times $t = 0, \dots, n$ (more details on this step later.)

2.2 Lattice approximation

The problem of automata fooling is closely linked to lattice approximation, defined as follows:

► **Definition 6** (Lattice Approximation Problem (LAP)). *Given an $m \times n$ matrix A and fractional vector $\vec{u} \in [0, 1]^n$, the objective is to compute an integral vector $\vec{v} \in \{0, 1\}^n$ to minimize the discrepancies $D_k = |\sum_{j=1}^n A_{kj}(u_j - v_j)|$.*

Intuitively, this models the process of “rounding” each random bit (represented by u_j) to a static zero or one (represented by v_j). The work [19] provides a parallel algorithm with near-optimal discrepancy as well as complexity; we summarize it as follows:

► **Theorem 7** (Theorem 1.3 of [19]). *Suppose that $A_{kj} \in [0, 1]$ for all k, j . There is a deterministic parallel algorithm for the LAP with $D_k \leq O(\sqrt{\mu_k \log m} + \log m)$ for all k , where $\mu_k = \sum_{j=1}^n A_{kj}u_j$. The algorithm uses $\tilde{O}(n + m + \text{nnz}(A))$ processors, where $\text{nnz}(A)$ denotes the number of non-zero entries in the matrix A .*

It will be convenient to allow for the discrepancy matrix A to take arbitrary real values.

► **Proposition 8.** *Let $\Delta_k = \max_j |A_{kj}|$ for each row k . There is a deterministic parallel algorithm for the LAP where $D_k \leq O(\sqrt{\Delta_k \tilde{\mu}_k \log m} + \Delta_k \log m) \leq O(\Delta_k(\sqrt{n \log m} + \log m))$ for all k , where $\tilde{\mu}_k = \sum_{j=1}^n |A_{kj}|u_j$. The algorithm uses $\tilde{O}(n + m + \text{nnz}(A))$ processors.*

Proof. Construct a $2m \times n$ matrix \tilde{A} , where for each row $k = 0, \dots, m-1$ of A , the matrix \tilde{A} has $\tilde{A}_{2k,j} = \max\{0, \frac{A_{kj}}{\Delta_k}\}$ and $\tilde{A}_{2k+1,j} = \max\{0, \frac{-A_{kj}}{\Delta_k}\}$. Then apply Theorem 7 to \tilde{A} . ◀

3 Overview of algorithms and data structures

The subroutine REDUCE (Section 4) is the technical core of the automata-fooling construction: given an input distribution E over drivestreams and a weight function w , it returns a distribution D which is close to E (measured in terms of w), but which has much smaller support. Importantly, through the use of appropriate data structures, the cost of REDUCE may be significantly less than the total size of E itself. Later, the final algorithm FOOL (Section 5) will be defined by repeated calls to REDUCE over successively larger time-horizons.

Concretely, we store each distribution D over window (t, h) as an array of drivestreams $D[0], \dots, D[\ell - 1] \in \Omega_{t, t+h}$ with associated probabilities $p_D(0), \dots, p_D(\ell - 1)$. Here $\ell = |D|$ is the *size* of D . By adding dummy zero entries, we can assume that $|D|$ is a power of two.

For a bitstring b of length at most $\lg |D|$, we denote by $D[b*]$ the induced distribution consisting of all the drivestreams in D whose indices start with b . So $D[b*] = D[b0*] \cup D[b1*]$, where $b0$ and $b1$ refer to concatenating a 1 or 0 bit to b . Similarly, $p_D(b*) = \sum_{a \in D[b*]} p_D(a)$.

We define the *Prediction Problem* for a distribution D and a weight function $w : S \rightarrow \mathbb{R}$ as follows: we need to produce a data structure $\mathcal{Q}(D, w)$, which can answer the following two types of queries: (i) given any bitstring b , return the value $p_D(b*)$; (ii) given (b, s) where b is a bitstring and s is a state, return the value $T_{D[b*]}(s, w)$. Each query should take polylogarithmic time and processors. In either case, b can take on any length $\ell \leq \lg |D|$.

► **Observation 9.** *Let D be a distribution on window (t, h) . The Prediction Problem for D and a weight function w can be solved with $\tilde{O}(|D|h\eta)$ processors.*

The most important case is for a Cartesian product, which will be used in Section 5. Critically, we can use the structure within a distribution to avoid materializing it explicitly.

► **Proposition 10.** *Given distributions D^1, D^2 on windows (t_1, h) and $(t_1 + h, h)$ respectively, the Prediction Problem for distribution $E = D^1 \times D^2$ and any weight function w can be solved with $\tilde{O}(\eta h(|D^1| + |D^2|))$ processors.*

4 The REDUCE Algorithm

To reiterate, the goal of REDUCE (Algorithm 1) is to take an input distribution E and weight function w , and produce a smaller distribution D which is close to it. Note that here w will not be the given weight function over final states W .

For intuition, consider the following process. Draw m elements $D[0], \dots, D[m - 1]$ randomly and independently with replacement from the support of the distribution E , wherein $D[i] = v$ is selected with probability proportional to $p_E(v)$. Then set $p_D[i] = 1/m$, so the process is unbiased. Via standard concentration bounds, appropriate choices of m ensure that $T_D(s, w) \approx T_E(s, w)$. The algorithm REDUCE is based on derandomizing this process via a slowed-down simulation: to compute D , we iteratively compute distributions $D_0, D_1, \dots, D_\ell = D$ by fixing the i^{th} bit level of each entry in D_{i-1} .

Algorithm 1 REDUCE(E, ε, w).

- 1: Set $\ell = \lg |E|$ and $m = \frac{C\ell^2 \log \eta}{\varepsilon^2}$ for a constant C .
 - 2: Solve Prediction Problem for distribution E
 - 3: Initialize the multiset $H_0 := \{m \text{ empty bitstrings}\}$
 - 4: **for** $i = 0, \dots, \ell - 1$ **do** ▷ Fix i^{th} bit
 - 5: Formulate LAP \mathcal{L} for H_i .
 - 6: $\vec{v} \leftarrow$ solve \mathcal{L} via Proposition 8 with sampling rate $u_b = \frac{p_E(b1*)}{p_E(b*)}$ for each $b \in H_i$
 - 7: $H_{i+1} \leftarrow \{bv_b : b \in H_i\}$ ▷ Concatenate vector \vec{v} as i^{th} bit level
 - 8: **parfor** $j \in \{0, \dots, m - 1\}$ **do** ▷ Convert bitstring (index) to drivestream (value)
 - 9: Set $D[j] = E[H_\ell[j]]$ and set probability $p_D[j] = 1/m$.
 - return** D
-

In order to measure distribution error, we introduce the following key definition:

► **Definition 11** (Sensitivity). *For a function $w : S \rightarrow \mathbb{R}$ and state s , the sensitivity $\alpha(s, w, E)$ is:*

$$\alpha(s, w, E) := \max_{\vec{r} \in E} w(F(\vec{r}, s)) - \min_{\vec{r} \in E} w(F(\vec{r}, s))$$

The following is our main result analyzing the algorithm:

► **Theorem 12.** *The algorithm REDUCE runs in $\tilde{O}((h + \eta)/\varepsilon^2)$ processors, plus the cost of solving the Prediction Problem for E . The final distribution D is uniform with size $O(\frac{\log \eta \log^2 |E|}{\varepsilon^2})$. For large enough constant C , the distribution D satisfies*

$$|T_D(s, w) - T_E(s, w)| \leq \varepsilon \alpha(s, w, E) \quad \text{for each state } s$$

Proof. Let us fix E, w ; for brevity, we write $\alpha_s := \alpha(s, w, E)$ for each state s .

Each multiset H_i contains m bitstrings $H_i[0], \dots, H_i[m - 1]$ of length i . Consider the distributions D_i obtained by drawing a bitstring $b \in H_i$ uniformly at random and then drawing drivestream from $E[b*]$. In particular, $D_0 = E$ and the distribution returned by REDUCE is precisely $D_\ell = D$. We have the following equation for every state s :

$$T_{D_i}(s, w) = \frac{1}{m} \sum_{b \in H_i} T_{E[b*]}(s, w).$$

Now, to analyze a given step i , observe that H_{i+1} is obtained by appending a bit v_b to each bitstring $b \in H_i$. We expose the choice of the next bit in D_i and D_{i+1} as follows

$$\begin{aligned} T_{D_i}(s, w) &= \frac{1}{m} \sum_{b \in H_i} \left[\frac{p_E(b1*)}{p_E(b*)} \cdot T_{E[b1*]}(s, w) + \frac{p_E(b0*)}{p_E(b*)} \cdot T_{E[b0*]}(s, w) \right] \\ T_{D_{i+1}}(s, w) &= \frac{1}{m} \sum_{b \in H_i} [v_b \cdot T_{E[b1*]}(s, w) + (1 - v_b) \cdot T_{E[b0*]}(s, w)]. \end{aligned}$$

Thus we can calculate the difference between probabilities for D_i and D_{i+1} as:

$$T_{D_{i+1}}(s, w) - T_{D_i}(s, w) = \frac{1}{m} \sum_{b \in H_i} \left(\frac{p_E(b1*)}{p_E(b*)} - v_b \right) (T_{E[b1*]}(s, w) - T_{E[b0*]}(s, w)). \quad (1)$$

In light of Eq. (1), we apply Proposition 8 where each state s corresponds to a constraint row k with entries $A_{kb} = T_{E[b1*]}(s, w) - T_{E[b0*]}(s, w)$, and with $u_b = \frac{p_E(b1*)}{p_E(b*)}$ for all b . It is evident that, after solving the Prediction Problem for E , the values $T_{E[bx*]}, p_E(bx*)$ for the Lattice Approximation Problem at Line 6 can be generated using $O(\eta m)$ processors. Furthermore, the maximum spread of the values $w(F(\vec{r}, s))$ over \vec{r} is at most α_s , so $\Delta_k = \max_b |A_{kb}| \leq \alpha_s$. Since the matrix has η rows and m columns, Proposition 8 gives:

$$|T_{D_{i+1}}(s, w) - T_{D_i}(s, w)| \leq O\left(\frac{\alpha_s \sqrt{m \log \eta} + \alpha_s \log \eta}{m}\right)$$

By our choice of m , this is at most $\varepsilon \alpha_s / \ell$. Over all iterations, this gives the desired bound

$$|T_D(s, w) - T_E(s, w)| = \sum_{i=0}^{\ell-1} |T_{D_{i+1}}(s, w) - T_{D_i}(s, w)| \leq \varepsilon \alpha_s. \quad \blacktriangleleft$$

5 The FOOL Algorithm

We build the automata-fooling distribution via the algorithm FOOL (Algorithm 2).

Algorithm 2 FOOL(ε) (Assume n is a power of two).

-
- 1: Set $D_{0,t} = \Omega_t$ for each $t = 0, \dots, n-1$
 - 2: Determine approximate transition vectors $\hat{V}_t(s) \approx T_{t,n}(s, W) : s \in S$ for all $t = 0, \dots, n$.
 - 3: **for** $i = 0, \dots, \lg n - 1$ **do**
 - 4: **parfor** $t \in \{0, 2h, 4h, 8h, n - 2h\}$ where $h = 2^i$ **do**
 - 5: $D_{i+1,t} \leftarrow \text{REDUCE}(D_{i,t} \times D_{i,t+h}, \delta, \hat{V}_{t+2h})$ for $\delta = \frac{\varepsilon}{20(1+\lg n)}$
 - return** final distribution $D = D_{\lg n, 0}$
-

The algorithm first finds the “expected value” vectors V_t for each distribution $\Omega_{t,n}$. This may take advantage of problem-specific automaton properties, and it may have some small error. We will also describe a few “generic” methods to calculate these probabilities exactly.

The main loop fools distributions on time horizons $h = 1, 2, 4, 8, \dots$ in a bottom-up fashion, and merges them together using the REDUCE procedure from the previous section. Specifically, at each level i , it executes REDUCE in parallel to combine $D_{i,t}$ and $D_{i,t+h}$ to obtain $D_{i+1,t}$. Note that we never materialize the Cartesian product of $D_{i,t}$ and $D_{i,t+h}$.

► **Theorem 13.** *The cost of FOOL is $\tilde{O}(n\eta/\varepsilon^2 + \eta\sigma)$ processors, plus the cost of computing the vectors $\hat{V}_t : t = 0, \dots, n$. The final distribution $D_{\lg n, 0}$ has size $O(\frac{\log^5(n\eta\sigma/\varepsilon)}{\varepsilon^2})$.*

Let us now proceed to analyze the error of the final distribution D . For purposes of analysis, we define the *exact* transition vector $V_t = T_{t,n}(s, W)$ and its approximation error by $\beta = \max_{s,t} |V_t(s) - \hat{V}_t(s)|$.

► **Theorem 14.** *For any state $s \in S$, the final distribution D returned by FOOL(ε, W) satisfies*

$$|T_D(s, W) - T_\Omega(s, W)| \leq \varepsilon \mathfrak{V}(s) + 3\beta n$$

Proof Sketch. For any state s and times $k \geq t$, let $a_{s,t,k}$ be the maximum value of $\alpha(s', V_{k+1}, \Omega_k)$ over $s' = F(\vec{r}, s) : \vec{r} \in \Omega_{t,k}$. We show by induction on i that each $D_{i,t}$ for any s satisfies

$$|T_{D_{i,t}}(s, V_{t+2^i}) - T_{t,t+2^i}(s, W)| \leq 3(2^i - 1)\beta + 2i\delta \sum_{k=t}^{t+2^i-1} a_{s,t,k} \quad (2)$$

The base case $i = 0$ is vacuous since $D_{0,t} = \Omega_t$. The final case $i = \lg n, t = 0$ establishes the claimed result, since $\delta = \frac{\varepsilon}{20(1+\lg n)}$ and $\mathfrak{V}(s) = \sum_{t=0}^{n-1} a_{s,0,t}$. ◀

As we have mentioned, there can be problem-specific shortcuts to compute (or approximate) the transition matrices T and resulting expected-value vectors V_t . There is the following generic method to compute them via matrix multiplication.

► **Proposition 15.** *All vectors $V_t = T_{t,n}(s, W)$, can be computed with $\tilde{O}(n\eta^\omega)$ processors, where ω is the exponent of any efficiently-parallelizable matrix-multiplication algorithm.*

In particular, for the Coppersmith-Winograd algorithm [16], we have $\omega \leq 2.38$. (See [29] for further details on parallel implementation.)

Moreover, if we can compute all transition matrices $T_{t,t+h} : h = 2^i, t = j2^i$, then we can compute all vectors $V_t = T_{t,n}(\cdot, W)$ exactly (for any W) with $\tilde{O}(n\eta^2)$ processors.

Therefore, when dealing with multiple statistical tests, we have the following result.

► **Corollary 16.** *Consider statistical tests $i = 1, \dots, \ell$, each computed by its own automaton F_i on a state space S_i of size $|S_i| = \eta_i$, with its own weight function W_i . When combined into a single automaton, the resulting automaton F has the following properties:*

1. *The total statespace is $\eta = \sum_i \eta_i$.*
2. *Each starting state s for automaton i has $\mathfrak{V}(s) = \mathfrak{V}_i(s)$, where \mathfrak{V}_i denotes the value of \mathfrak{V} for weight function W_i and automaton F_i .*
3. *The exact vectors V_t can be calculated in $\tilde{O}(n \sum_i \eta_i^\omega)$ processors.*

In particular, the FOOL algorithm has processor complexity $\tilde{O}(n\eta/\varepsilon^2 + \sigma\eta + n \sum_i \eta_i^\omega)$, and the resulting distribution D has $|T_D(s, W) - T_\Omega(s, W)| \leq \varepsilon \mathfrak{V}_i(s)$ for each state s of automaton i .

6 Reducing the state space for counter automata

Given a weight function W , let us consider a statistic of the form

$$W\left(\sum_t f_t(r_t)\right) \quad \text{for functions } f_t : \Omega_t \rightarrow \mathbb{Z}$$

We refer to such statistics as *counters*. They are ubiquitous in randomized algorithm; they are often used directly, and can also be combined together into higher-order statistics.

In the automata-fooling setting, it is easy to compute $\sum_t f_t(r_t)$ by tracking all possible values for the running sum. However, this is often overkill: typically, the running sum is confined to a much smaller window, roughly the standard deviation around the mean. We can take advantage of this by constructing an automaton which only maintains the running sum within “typical” values close to the mean, along with a “reject” state for some exponentially-rare deviations. Let us define some relevant parameters for the counter.

$$\mu_t = \mathbb{E}_{r \sim \Omega_t}[f_t(r)], \quad M_t = \max_{r \in \Omega_t} |f_t(r) - \mu_t|, \quad \kappa = \sum_{t=0}^{n-1} \mathbb{V}_{r \sim \Omega_t}[f_t(r)], \quad M = \max_{t \in [n]} M_t$$

Given some error parameter $\delta > 0$, we choose a value B with $B \geq \lceil 100(1 + M + \sqrt{\kappa}) \log(n/\delta) \rceil$. In this context, we refer to B as the *span* of the automaton.

► **Observation 17.** *For a drivestream \vec{r} drawn randomly from Ω , it holds with probability at least $1 - \delta$ that $|\sum_{i=t}^{t'} (f(r_i) - \mu_i)| < 0.15B$ for all times t, t' .*

Proof. Apply Bernstein’s inequality and take a union bound over t, t' . ◀

In light of Observation 17, our strategy will be to construct a truncated automaton \tilde{F} , which only stores the running sum within a window of $\pm B$ from the running mean. Define $a_t = \lceil \sum_{i=0}^t \mu_i \rceil$ for each value t . The truncated automaton has state space $\tilde{S} = \{-B, -B+1, \dots, B-1, B\} \cup \{\perp\}$; given input r_t , it updates its state c to a new state c' as:

$$c' = \begin{cases} c + f(r_t) - a_t + a_{t-1} & \text{if } c \neq \perp \text{ and } |c + f(r_t) - a_t + a_{t-1}| \leq B \\ \perp & \text{if } c = \perp \text{ or } |c + f(r_t) - a_t + a_{t-1}| > B \end{cases}$$

Each integer state c at time t corresponds to a running sum $c + a_t$. The state \perp represents a “reject” or “rare” state where the running sum has deviated too far from its mean. We define a related potential function $\phi : \tilde{S} \rightarrow [0, 1]$ as

$$\phi(c) = \begin{cases} 1 & \text{if } |c| \leq B/3 \\ (2B/3 - |c|)/(B/3) & \text{if } B/3 < |c| < 2B/3 \\ 0 & \text{if } |c| > 2B/3 \text{ or } c = \perp \end{cases}$$

Intuitively, ϕ measures how close the current state c is to a reject state. It can be used to “damp” the weight function W and make a gradual transition to the reject state.

For purposes of analysis, it is useful to compare \tilde{F} with the “truthful” automaton F . The potential function ϕ can also be applied on the original state space S , where we use the convention that any states $|s| > B$ corresponds to the reject state \perp of \tilde{S} .

Relevant properties of \tilde{F} such as computing its transition matrix or its weight variability can be derived from F up to a small error. Indeed, our application to Gale-Berlekamp Switching Game will use precisely this approach. For the MAX-CUT application, we will need to combine multiple truncated automata; this will require a slightly different damping function and analysis.

► **Proposition 18.** *Let W be a weight function for automaton F and define*

$$\tilde{W}(c) = \begin{cases} 0 & \text{if } c = \perp \\ \phi(c)W(c) & \text{if } c \neq \perp, \end{cases} \quad \Delta = \max_{c \in \tilde{S}} |W(c)|, \quad \tilde{\Delta} = \max_{\substack{c \in \tilde{S}: \\ |c - \mu_t| \leq 2B}} |W(c)|.$$

Let T, \tilde{T} denote the transition matrices for automata F, \tilde{F} respectively.

1. *For any time t and state $c \in S$, there holds $|\tilde{T}_{t,n}(c, \tilde{W}) - T_{t,n}(c, \tilde{W})| \leq \delta \Delta$.*
2. *For the starting state $c = 0$ and time $t = 0$, there holds $|\tilde{T}_{t,n}(c, \tilde{W}) - T_{t,n}(c, W)| \leq \delta \Delta$.*
3. *For any time t and state c , there holds*

$$\tilde{\mathfrak{C}}(c, t) \leq \mathfrak{L}(c, t) + 2\delta\Delta + 6\tilde{\Delta}M_t/B,$$

where $\tilde{\mathfrak{C}}$ denotes the confusion with respect to automata \tilde{F} and weight function \tilde{W} , while \mathfrak{L} denotes the Lipschitz value with respect to automaton F and weight function W .

Note that, in both these cases, we use the convention that if $|c| > B$, then c corresponds to the reject state of automaton \tilde{F} .

7 Applications

We present the derandomization of two basic problems: the Gale-Berlekamp Switching Game (Section 7.1) and approximate MAX-CUT via SDP rounding (Section 7.2). Through our new construction, we improve the deterministic processor complexity for both problems.

7.1 Gale-Berlekamp Switching Game

The Gale-Berlekamp Switching Game is a classic combinatorial problem: given an $n \times n$ matrix A with entries $A_{ij} = \pm 1$, we want to find vectors $x, y \in \{-1, +1\}^n$ to maximize the imbalance $I = \sum_{i,j} A_{i,j} x_i y_j$. An elegant randomized algorithm of [4], based on the Central Limit Theorem, gives imbalance of $I \geq (\sqrt{2/\pi} - o(1))n^{3/2}$. This was derandomized in [24], using automata-fooling with a processor complexity of $n^{5+o(1)}$.

► **Theorem 19.** *There is a parallel deterministic algorithm to find $\vec{x}, \vec{y} \in \{-1, +1\}^n$ with imbalance $I \geq (\sqrt{2/\pi} - o(1))n^{3/2}$ using $\tilde{O}(n^{3.5})$ processors.*

In order to show Theorem 19, following [8] and [4], we set $x_i = 1$ if $\sum_j A_{i,j} y_j > 0$, and $x_i = -1$ otherwise. This gives

$$\sum_{i,j} A_{i,j} x_i y_j = \sum_i x_i \sum_j A_{i,j} y_j = \sum_i \left| \sum_j A_{i,j} y_j \right|.$$

As shown in [13], for \vec{y} uniformly drawn from $\Omega = \{-1, +1\}^n$, we have $\mathbb{E}[\left| \sum_j A_{i,j} y_j \right|] \geq \sqrt{2n/\pi} - o(\sqrt{n})$ for each i . This is the statistical test we want to fool.

We will choose our drivestream values to be $r_t = y_t$, with Ω_t the uniform distribution on $\{-1, 1\}$. For each value i , we have a truncated counter automaton \tilde{F}_i to track $\sum_j A_{i,j} r_j$. This uses the construction of Section 6 with $\delta = n^{-10}$, where $M = 1$ and $\kappa = \sum_j A_{i,j}^2 = n$ and $B = \tilde{O}(\sqrt{n})$. This automaton uses the weight function

$$\tilde{W}(c) = \begin{cases} 0 & \text{if } c = \perp \\ \phi(c)|c| & \text{if } c \neq \perp \end{cases}$$

► **Proposition 20.** *Fix a row i and automaton $\tilde{F} = \tilde{F}_i$, and correspondingly let F be the full “truthful” automaton. Let T, \tilde{T} be the transition matrices for F, \tilde{F} respectively.*

1. *For any time t and state s of F , we have $|T_{t,n}(s, \tilde{W}) - \tilde{T}_{t,n}(s, \tilde{W})| \leq 2\delta n$.*
2. *The final state $\tilde{z} = \tilde{F}(\vec{y}, 0)$ for $\vec{r} \sim \Omega$ satisfies $\mathbb{E}[\tilde{W}(\tilde{z})] \geq \sqrt{2n/\pi} - o(\sqrt{n})$.*
3. *The weight function \tilde{W} for automaton \tilde{F} has total variability $O(n)$.*

Proof. We apply here Proposition 18 with $W = |c|$. Clearly here $\mathfrak{L}(c, t) = 2$ and $M_t \leq 1$ for all c, t . Also, we calculate $\Delta = n, \tilde{\Delta} \leq \tilde{O}(\sqrt{n})$. With Proposition 18(a), this proves (a). Similarly, by Proposition 18(b), we have

$$\mathbb{E}[\tilde{W}(\tilde{z})] \leq \delta n + \mathbb{E}_\Omega[W(z)] \leq o(\sqrt{n}) + (\sqrt{2/\pi} - o(1)) \cdot \sqrt{n}$$

where z is the final state for the automaton F .

Finally, for (c), we use the definition of total variability and Proposition 18(c) to get:

$$\mathfrak{V}(c) = \sum_{t=0}^{n-1} \max_{\vec{r} \in \Omega_{0,t}} \tilde{\mathfrak{C}}(\tilde{F}(\vec{r}, c), t) \leq \sum_{t=0}^{n-1} (\mathfrak{L}(c, t) + 2\Delta\delta + 6\tilde{\Delta}M_t/B) = \tilde{O}(n). \quad \blacktriangleleft$$

The complexity now follows from Corollary 16 with $\varepsilon = 1/\sqrt{n \log n}$, where we have n automata each of state space $\hat{\eta}_i = \tilde{O}(\sqrt{n})$. By applying Proposition 20, we get that for any sequence \vec{y} drawn from the resulting distribution D and automaton \tilde{F}_i ,

$$\begin{aligned} |\tilde{T}_D(0, \tilde{W}) - T_\Omega(0, W)| &\leq |\tilde{T}_D(0, \tilde{W}) - \tilde{T}_\Omega(0, \tilde{W})| + |\tilde{T}_\Omega(0, \tilde{W}) - T_\Omega(0, W)| \\ &\leq \varepsilon \mathfrak{V}(0) + o(\sqrt{n}) = o(\sqrt{n}) \end{aligned}$$

Therefore, for any row i , the final state \tilde{z} satisfies

$$\mathbb{E}_D \left[\left| \sum_j A_{ij} y_j \right| \right] \geq \mathbb{E}_\Omega [W_i(\tilde{z})] - o(\sqrt{n}) \geq (\sqrt{2/\pi} - O(\epsilon))\sqrt{n}$$

By searching the space exhaustively, we can find a specific sequence \vec{y} satisfying the desired imbalance bounds. This concludes the proof of Theorem 19.

7.2 Approximate MAX-CUT

The MAX-CUT problem for a graph $G = (V, E)$ is to find a vertex set $S \subseteq V$ to maximize the total weight of edges with exactly one endpoint in S . The seminal work of Goemans & Williamson [21] showed that MAX-CUT can be approximated to a factor of $\alpha \approx 0.878$ by rounding its semi-definite programming (SDP) formulation. Moreover, the integrality gap of the SDP is precisely α [18], and assuming the Unique Games Conjecture no better approximation ratio is possible for polynomial-time algorithms [34].

A sequential deterministic α -approximation with $\tilde{O}(n^3)$ runtime was shown in [9]. This relies on the method of conditional expectation, which is hard to parallelize. The work of [43] used the automata derandomization framework to get a parallel deterministic $\alpha(1 - \epsilon)$ -approximation algorithm. The main downside of this approach is the huge polynomial processor complexity (on the order of n^{100}). We give an improved analysis with our new automata-fooling construction and some other optimizations.

To review, note that MAX-CUT can be formulated as the following integer program:

$$\max \quad \frac{1}{2} \sum_{(i,j)=e \in E} w_e(1 - v_i \cdot v_j) \quad \text{s.t.} \quad v_i \in \{-1, 1\}, i \in [n].$$

This can be relaxed to the following SDP:

$$\max \quad \frac{1}{2} \sum_{(i,j)=e \in E} w_e(1 - v_i \bullet v_j) \quad \text{s.t.} \quad v_i \in [-1, 1]^n, \|v_i\|_2 = 1, i \in [n],$$

where \bullet denotes the inner product in \mathbb{R}^n .

We can round a given SDP solution v by drawing independent standard Gaussian random variables X_0, \dots, X_{n-1} and constructing the cut $S = \{i \mid v_i \bullet X \geq 0\}$, which gives cutsizes W

$$\begin{aligned} \mathbb{E}[W] &= \sum_{(i,j)=e \in E} w_e \Pr[(i, j) \text{ is cut}] = \sum_{(i,j)=e \in E} w_e \Pr[\text{sgn}(v_i \bullet X) \neq \text{sgn}(v_j \bullet X)] \\ &= \sum_{(i,j)=e \in E} w_e \frac{\arccos(v_i \bullet v_j)}{\pi} \geq \alpha \cdot \text{OPT}, \end{aligned}$$

where OPT denotes the size of the maximum cut. Following [43], we will derandomize this by fooling each statistical test $\Pr[\text{sgn}(v_i \bullet X) \neq \text{sgn}(v_j \bullet X)]$.

► **Theorem 21.** *There is a deterministic parallel algorithm to find an $\alpha(1 - \epsilon)$ -approximate rounding of a MAX-CUT SDP solution with $\tilde{O}(\frac{mn^3}{\epsilon^4})$ processors.*

Via the algorithm of [1] to solve the SDP, this gives the following main result:

► **Corollary 22.** *For arbitrary constant $\epsilon > 0$, we can find an $\alpha(1 - \epsilon)$ -approximate MAX-CUT solution using $\tilde{O}(mn^3)$ processors and polylogarithmic time.*

The remainder of the section is devoted to showing Theorem 21. To avoid cumbersome calculations, we will show an algorithm for a $(1 - O(\varepsilon))$ -approximation, assuming ε is smaller than any needed constants. For readability, $\tilde{O}()$ suppresses any $\text{polylog}(n/\varepsilon)$ terms.

As a starting point, consider the following process. We draw n independent standard Gaussian variables X_0, \dots, X_{n-1} . For each edge (i, j) , we have a statistical test to read these values in order and compute $c_i = v_i \bullet X$ and $c_j = v_j \bullet X$. We then apply the weight function $W_{ij}(c_i, c_j)$ as the indicator variable that $\text{sgn}(c_i) = \text{sgn}(c_j)$.

In order to apply our derandomization framework efficiently, we make a number of changes to this basic strategy: (i) we define the drivestream value r_t to be a discretized truncated Gaussian; (ii) we further quantize each term $v_{ik}r_k$ in the computation of the sum $c_i = \sum_k v_{ik}r_k$; (iii) we apply the optimization of Section 6 to reduce the state space for each counter automaton; and (iv) we modify the weight function to a pessimistic estimator with better Lipschitz properties. Let us fix two quantization parameters for some constant $C > 0$

$$\gamma = \frac{\varepsilon}{C\sqrt{n \log(n/\varepsilon)}}, \quad R = C \log(n/\varepsilon)$$

Concretely, each drivestream value r_k is derived by truncating a standard Gaussian to within $\pm R$, and rounding to the nearest multiple of γ . Then, each term in the product $v_{ik}r_k$ is rounded to the nearest multiple of γ . Since this comes up frequently in the analysis, let us denote this “rounded” inner product by: $v \star r = \gamma \sum_k \lceil \frac{1}{\gamma} v_k r_k \rceil$. Given this definition, we will form our cut set by setting $S = \{i : v_i \star r \geq 0\}$.

► **Lemma 23.** *For appropriately chosen C , there is a coupling between random variables X and Y such that, for any unit vector $u \in \mathbb{R}^n$, there holds $\Pr(|u \bullet X - u \star r| > \varepsilon) \leq (\varepsilon/n)^{10}$.*

Observe that the scaled sum $\frac{1}{\gamma}(v_i \star r)$ is an integer-valued counter. We can apply the method of Section 6 to construct a “vertex” automaton \tilde{F}_i that tracks the running sum within a window. We record a few parameters and observations about this automaton.

► **Proposition 24.** *The automaton \tilde{F}_i can be implemented with the following parameters:*

$$M_t \leq \tilde{O}(|v_{it}|/\gamma), \quad M \leq \tilde{O}(1/\gamma), \quad \kappa \leq \tilde{O}(1/\gamma^2), \quad \delta = (\varepsilon/n)^{10}, \quad B = \tilde{\Theta}(1/\gamma).$$

From these vertex-automata, we construct an edge-automaton \tilde{F}_{ij} for each edge $(i, j) \in E$. Automaton \tilde{F}_{ij} keeps track of the joint states of \tilde{F}_i, \tilde{F}_j , i.e. the truncated running sums $v_i \star r$ and $v_j \star r$. It uses the following weight function for the state $s = (c_i, c_j)$:

$$\tilde{W}_{ij}(s) = \begin{cases} 0 & \text{if } c_i = \perp \text{ or } c_j = \perp \\ 0 & \text{if } \text{sgn}(c_i) = \text{sgn}(c_j) \\ \min\{1, \frac{|c_i - c_j|}{\varepsilon}\} \phi_i(c_i) \phi_j(c_j) & \text{otherwise} \end{cases}$$

Note that whenever $\tilde{W}_{ij}(s) > 0$, we have $\text{sgn}(v_i \star r) \neq \text{sgn}(v_j \star r)$, i.e. edge ij is cut.

► **Proposition 25.** *For an edge ij , let $\tilde{F} = \tilde{F}_{ij}$, and consider the “truthful” automaton $F = F_{ij}$ (which uses the full untruncated automata for vertices i and j). Let T, \tilde{T} denote the transition matrices for F, \tilde{F} respectively.*

1. *For any time t and state s of F , there holds $|T_{t,n}(s, \tilde{W}) - \tilde{T}_{t,n}(s, \tilde{W})| \leq 2\delta$.*
2. *The final automaton state $\tilde{z} = \tilde{F}(\tilde{r}, 0)$ for $\tilde{r} \sim \Omega$ satisfies $\mathbb{E}[\tilde{W}(\tilde{z})] \geq \frac{\arccos(v_i \bullet v_j)}{\pi} - O(\varepsilon)$.*
3. *For drivestreams \tilde{r}^1, \tilde{r}^2 differing in coordinate t , the final states $z^1 = F(\tilde{r}^1, 0), z^2 = F(\tilde{r}^2, 0)$ of automaton F have $|\tilde{W}(z^1) - \tilde{W}(z^2)| \leq \tilde{O}(\varepsilon^{-1}(|v_{it}| + |v_{jt}|))$.*
4. *The weight function \tilde{W} has total variability $\tilde{O}(\sqrt{n}/\varepsilon)$ for automaton \tilde{F} .*

Proof. The proof is very similar to Proposition 18; see full version. ◀

Next, we discuss how to approximate the transition matrices for the automata. In light of Proposition 25(a), we will compute vectors $\hat{V}_t(s) = T_{t,n}(s, W)$ for the full automaton F_{ij} . This is much easier than computing the transition matrices for \tilde{F}_{ij} , since we do not need to track reject states. In particular, this achieves error $\beta = 2\delta$ compared to $V_t = \tilde{T}_{t,n}(s, W)$.

► **Proposition 26.** *For each automaton F_{ij} , the vectors $T_{t,n}(s, W) : t = 0, \dots, n$, can be computed with $\tilde{O}(n^3 \varepsilon^{-4})$ processors.*

Proof Sketch. We will recursively compute transition matrices $T_{t,h}$ for $h = 1, 2, 4, \dots$, and t divisible by h . The critical observation here is that since F_{ij} is a pair of counters, we only need to keep track of the probability distribution on the difference pairs

$$\left(\sum_{k=t}^{t+h-1} \lceil v_{ik} r_k / \gamma \rceil, \sum_{k=t}^{t+h-1} \lceil v_{jk} r_k / \gamma \rceil \right).$$

This is effectively a convolution, computable via an FFT. ◀

We are now ready to compute the total complexity. Each probability distribution Ω_t has size $\tilde{O}(\gamma^{-1}) = \tilde{O}(\sqrt{n}/\varepsilon)$, thus $\sigma = \tilde{O}(n^{1.5}/\varepsilon)$. The reduced total number of states of an edge-automaton F_{ij} is $\tilde{O}(1/\gamma^2) = \tilde{O}(n/\varepsilon^2)$ and the weight function W_{ij} has total variability $\tilde{O}(\sqrt{n}/\varepsilon)$. So we run FOOL with parameter $\varepsilon' = \varepsilon/\sqrt{n}$, and the fooling process has cost $\tilde{O}(m \frac{n^3}{\varepsilon^4} + m \frac{n^{2.5}}{\varepsilon^3})$. Consider the resulting distribution D . The edge i, j is only cut if $\tilde{W}_{ij}(s) > 0$. For each F_{ij} , by Theorem 14, the resulting final state z satisfies

$$\Pr_{\tilde{r} \sim D}(\text{edge } ij \text{ cut}) \geq \mathbb{E}_{\tilde{r} \sim D}[W_{ij}] \geq \mathbb{E}_{\tilde{r} \sim \Omega}[W_{ij}] - O(\varepsilon) - O(\beta n)$$

Here Proposition 25(a) gives $\beta \leq 2\delta \ll \varepsilon/n$, and Proposition 25(b) gives $\mathbb{E}_{\tilde{r} \sim \Omega} \geq \frac{\arccos(v_i \bullet v_j)}{\pi} - O(\varepsilon)$. Thus, overall, the edge i, j is cut with prob. at least $\frac{\arccos(v_i \bullet v_j)}{\pi} - O(\varepsilon)$. Summing over all edges, the total expected weight of the cut edges is

$$\sum_{ij \in E} w_e \frac{\arccos(v_i \bullet v_j)}{\pi} - \sum_{ij \in E} w_e O(\varepsilon)$$

The first term is at least $\alpha \cdot \text{OPT}$. The second term is at most $O(\varepsilon \cdot \text{OPT})$ since $\text{OPT} \geq \sum_e w_e/2$. By searching D exhaustively, we can find a cut satisfying these bounds.

References

- 1 Zeyuan Allen-Zhu, Yin Tat Lee, and Lorenzo Orecchia. Using optimization to obtain a width-independent, parallel, simpler, and faster positive SDP solver. In *Proc. 27th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1824–1831, 2016. doi:10.1137/1.9781611974331.ch127.
- 2 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.
- 3 Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k -wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992. doi:10.1002/rsa.3240030308.
- 4 Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2015. URL: <http://cs.nyu.edu/cs/faculty/spencer/nogabook/nogabook.html>.

- 5 Yossi Azar, Rajeev Motwani, and Joseph Naor. Approximating probability distributions using small sample spaces. *Combinatorica*, 18(2):151–171, 1998. doi:10.1007/PL00009813.
- 6 Nikhil Bansal, Nitish Korula, Viswanath Nagarajan, and Aravind Srinivasan. Solving packing integer programs via randomized rounding with alterations. *Theory of Computing*, 8(24):533–565, 2012. doi:10.4086/toc.2012.v008a024.
- 7 Mihir Bellare and John Rompel. Randomness-efficient oblivious sampling. In *Proc. 35th annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–287, 1994. doi:10.1109/SFCS.1994.365687.
- 8 Bonnie Berger. The fourth moment method. *SIAM Journal on Computing*, 26(4):1188–1207, 1997. doi:10.1137/S0097539792240005.
- 9 Ankur Bhargava and S Rao Kosaraju. Derandomization of dimensionality reduction and SDP based algorithms. In *Workshop on Algorithms and Data Structures*, pages 396–408. Springer, 2005. doi:10.1007/11534273_35.
- 10 Manuel Blum and Oded Goldreich. Towards a computational theory of statistical tests. In *Proc. 33rd annual Symposium on Foundations of Computer Science (FOCS)*, pages 406–416, 1992. doi:10.1109/SFCS.1992.267749.
- 11 Allan Borodin. On relating time and space to size and depth. *SIAM journal on computing*, 6(4):733–744, 1977. doi:10.1137/0206054.
- 12 Mark Braverman, Anup Rao, Ran Raz, and Amir Yehudayoff. Pseudorandom generators for regular branching programs. *SIAM Journal on Computing*, 43(3):973–986, 2014. doi:10.1137/120875673.
- 13 Thomas A Brown and Joel H Spencer. Minimization of ± 1 matrices under line shifts. In *Colloquium Mathematicum*, volume 1, pages 165–171, 1971.
- 14 Suresh Chari, Pankaj Rohatgi, and Aravind Srinivasan. Improved algorithms via approximations of probability distributions. In *Proc. 26th annual ACM Symposium on Theory of Computing (STOC)*, pages 584–592, 1994. doi:10.1145/195058.195411.
- 15 Lijie Chen, William M. Hoza, Xin Lyu, Avishay Tal, and Hongxun Wu. Weighted pseudorandom generators via inverse analysis of random walks and shortcutting. In *Proc. 64th annual Symposium on Foundations of Computer Science (FOCS)*, pages 1224–1239, 2023. doi:10.1109/FOCS57990.2023.00072.
- 16 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proc. 19th annual ACM Symposium on Theory of Computing (STOC)*, pages 1–6, 1987. doi:10.1145/28395.28396.
- 17 Guy Even, Oded Goldreich, Michael Luby, Noam Nisan, and Boban Veličkovic. Approximations of general independent distributions. In *Proc. 24th annual ACM Symposium on Theory of Computing (STOC)*, pages 10–16, 1992. doi:10.1145/129712.129714.
- 18 Uriel Feige and Gideon Schechtman. On the optimality of the random hyperplane rounding technique for MAX CUT. *Random Structures & Algorithms*, 20(3):403–440, 2002. doi:10.1002/rsa.10036.
- 19 Mohsen Ghaffari and Christoph Grunau. Work-efficient parallel derandomization II: Optimal concentrations via bootstrapping. In *Proc. 56th annual ACM Symposium on Theory of Computing (STOC)*, pages 1889–1900, 2024. doi:10.1145/3618260.3649668.
- 20 Mohsen Ghaffari, Christoph Grunau, and Václav Rozhoň. Work-efficient parallel derandomization I: Chernoff-like concentrations via pairwise independence. In *Proc. 64th annual Symposium on Foundations of Computer Science (FOCS)*, pages 1551–1562, 2023. doi:10.1109/FOCS57990.2023.00094.
- 21 Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995. doi:10.1145/227683.227684.
- 22 Parikshit Gopalan, Raghu Meka, Omer Reingold, Luca Trevisan, and Salil Vadhan. Better pseudorandom generators from milder pseudorandom restrictions. In *Proc. 53rd annual Symposium on Foundations of Computer Science (FOCS)*, pages 120–129, 2012. doi:10.1109/FOCS.2012.77.

- 23 Parikshit Gopalan, Raghu Meka, Omer Reingold, and David Zuckerman. Pseudorandom generators for combinatorial shapes. In *Proc. 43rd annual ACM Symposium on Theory of Computing (STOC)*, pages 253–262, 2011. doi:10.1145/1993636.1993671.
- 24 David G Harris. Deterministic parallel algorithms for bilinear objective functions. *Algorithmica*, 81:1288–1318, 2019. doi:10.1007/s00453-018-0471-0.
- 25 David G Harris. Deterministic algorithms for the Lovász local lemma: simpler, more general, and more parallel. *Random Structures & Algorithms*, 63(3):716–752, 2023. doi:10.1002/rsa.21152.
- 26 David G Harris, Ehab Morsy, Gopal Pandurangan, Peter Robinson, and Aravind Srinivasan. Efficient computation of sparse structures. *Random Structures & Algorithms*, 49(2):322–344, 2016. doi:10.1002/rsa.20653.
- 27 William M Hoza, Edward Pyne, and Salil Vadhan. Pseudorandom generators for unbounded-width permutation branching programs. In *Proc. 12th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 185, pages 7:1–7:20, 2021. doi:10.4230/LIPIcs.ITCS.2021.7.
- 28 Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 356–364, 1994. doi:10.1145/195058.195190.
- 29 Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley Publishing Company, 1992.
- 30 Anatole Joffe. On a set of almost deterministic k -independent random variables. *the Annals of Probability*, 2(1):161–162, 1974.
- 31 David R Karger and Daphne Koller. (De)randomized construction of small sample spaces in NC. *Journal of Computer and System Sciences*, 55(3):402–413, 1997. doi:10.1006/jcss.1997.1532.
- 32 Howard Karloff and Yishay Mansour. On construction of k -wise independent random variables. In *Proc. 26th annual ACM Symposium on Theory of Computing (STOC)*, pages 564–573, 1994. doi:10.1007/BF01196134.
- 33 Zander Kelley and Raghu Meka. Random restrictions and PRGs for PTFs in Gaussian space. In *Proc. 37th Computational Complexity Conference (CCC)*, 2022. doi:10.4230/LIPIcs.CCC.2022.21.
- 34 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O’Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007. doi:10.1137/S0097539705447372.
- 35 Michael Luby. A simple parallel algorithm for the maximal independent set problem. In *Proc. 17th annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1985. doi:10.1145/22145.22146.
- 36 Sanjeev Mahajan, Edgar A Ramos, and KV Subrahmanyam. Solving some discrepancy problems in NC. *Algorithmica*, 29(3):371–395, 2001. doi:10.1007/s004530010046.
- 37 Raghu Meka, Omer Reingold, and Avishay Tal. Pseudorandom generators for width-3 branching programs. In *Proc. 51st annual ACM Symposium on Theory of Computing (STOC)*, pages 626–637, 2019. doi:10.1145/3313276.3316319.
- 38 Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. In *Proc. 22nd annual ACM Symposium on Theory of Computing (STOC)*, pages 213–223, 1990. doi:10.1145/100216.100244.
- 39 Noam Nisan. Pseudorandom generators for space-bounded computations. In *Proc. 22nd annual ACM Symposium on Theory of Computing (STOC)*, pages 204–212, 1990. doi:10.1145/100216.100242.
- 40 Noam Nisan. $RL \subseteq SC$. In *Proc. 24th annual ACM Symposium on Theory of Computing (STOC)*, pages 619–623, 1992. doi:10.1145/129712.129772.

- 41 Ryan O'Donnell, Rocco A Servedio, and Li-Yang Tan. Fooling Gaussian PTFs via local hyperconcentration. In *Proc. 52nd annual ACM Symposium on Theory of Computing (STOC)*, pages 1170–1183, 2020. doi:10.1145/3357713.3384281.
- 42 Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff–Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995. doi:10.1137/S089548019223872X.
- 43 D Sivakumar. Algorithmic derandomization via complexity theory. In *Proc. 34th annual ACM Symposium on Theory of Computing (STOC)*, pages 619–626, 2002. doi:10.1145/509907.509996.
- 44 Aravind Srinivasan. New approaches to covering and packing problems. In *Proc. 12th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 567–576, 2001. URL: <http://dl.acm.org/citation.cfm?id=365411.365535>.